KARELIA-AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma

Heikki Kolehmainen

ARKISTOINTITIETOJÄRJESTELMÄN KEHITYSSUUNITELMA

Opinnäytetyö
Joulukuu 2016

Karelia AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Joulukuu 2016
Tietojenkäsittelyn koulutusohjelma

Karjalankatu 3
80200 JOENSUU
(013) 260 600

Tekijä(t)
Heikki Kolehmainen

Nimeke
Arkistointitietojärjestelmän kehityssuunnitelma

Toimeksiantaja
Karelia-ammattikorkeakoulu

Tiivistelmä

Opinnäytetyön tavoitteena oli tarkastella toimeksiantajan arkistointitietojärjestelmää ja löytää siihen uusi toteutustapa. Samalla kartoitettiin nykyisen tietojärjestelmän puutteita ja mahdollisia kehityskohteita. Opinnäytetyössä käsitellään myös verkkosovelluskehityksen uusia teknologioita sekä tutustutaan hieman verkkosovelluskehityksen historiaan.

Nykyisen tietojärjestelmän kehityskohteiden arvioinnissa ei käytetty varsinaisia mittareita, koska sopivia mittareita verkkosovelluksen nykyisen toiminnan analysoimiseksi ei ollut. Työssä pohdittiin, millainen hyvän tietojärjestelmän tulisi olla. Nykyisen tietojärjestelmän kehityskohteita ja puutteita selvitettiin vertaamalla tietojärjestelmää näihin vaatimuksiin. Työssä tutustuttiin myös Node.js-, Play! Framework- ja CakePHP-verkkosovelluskehityksiin ja tutkittiin, voiko verkkosovellusten avulla saada tietojärjestelmä reaaliaikaisemmaksi, tietoturvallisemmaksi ja käyttäjäystävällisemmäksi.

Opinnäytetyön tuloksena löytyi kaksi varteenotettavaa verkkosovelluskehystä Play! Framework ja Node.js, joiden avulla nykyinen arkistointitietojärjestelmä voidaan kehittää ja rakentaa modernin verkkosovelluskehityksen vaatimusten mukaiseksi. Edellä mainitut verkkosovelluskehykset tarjoavat myös monipuoliset ratkaisut opinnäytetyössä esitettyihin tietojärjestelmän kehityskohteisiin.

Kieli	Sivuja 44
suomi	Liitteet

Asiasanat

verkkosovelluskehitys, sovelluskehykset, tietojärjestelmät

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
November 2016
All Degree Programmes

Karjalankatu 3
80200 JOENSUU
FINLAND
(013) 260 600

Author (s)

Heikki Kolehmainen

Title
Development Plan for an Archiving Information System

Commissioned by
KUAS

Abstract

The main goal of this thesis was to examine the commissioner's information archiving system and to find a new method for implementation. At the same time, we research existing information and development gaps of the current information archiving system were mapped out.

The thesis also covers current and new technologies in web application development as well as some history. In the evaluation of the development targets no actual process metrics were used because no suitable indicators to analyze the web application was found. In this thesis it also discussed, what is a good information system like. The development areas and gaps of the current information system were compared to the requirements of a good information system.

The thesis also explored Node.js-, Play! Framework- and CakePHP's web application frameworks, and examined whether the apps on the web would contribute to better information retrieval in real time, data security and user-friendliness.

As a result, two respectable web application frameworks were found, that is, Play! Framework and Node.js which allow further development and building of the current information archiving system according to the requirements of modern web application development. The above-mentioned web application frameworks also offer a wide range of solutions for the development gaps presented in the thesis.

Language	Pages 44
Finnish	Appendices

Keywords

Web Application Development, application frameworks, information systems

Sisältö

Lyhenteet

HTML Hyper Text Markup Language on verkkosivulla käytettävä merkintäkieli. Sillä määritellään verkkosivuston rakenne ja merkistön (Korpela & Linjama 2005, 70).

DOM Document Object Model on oliopohjainen HTML-verkkosivun rakennemalli, jonka avulla dynaamista tietoa kohdistetaan HTML-sivulla (Korpela 2011).

WWW World Wide Web on yleiskäsite HTML-dokumenteille, joka toimii internetin välityksellä (Korpela j& Linjama 2005, 406).

Ajax Asynchronous JavaScript And XML on asynkroninen tiedonvälittystekniikka. Sen avulla sivuston dynaamista sisältöä voi muokata (Smith ja Negrino 2008, 359).

CSS Cascading Style Sheets on www-dokumenteissa käytetty tyylikirjasto. Sen parametrit määrittävät www-dokumentin visuaalisen näkymän tai ulkoasun (Korpela 2013).

JSON JavaScript Object Notation on avoin standardi, jonka avulla voidaan välittää tietoa tietojärjestelmien välillä (W3schools 2016).

JQuery Avoimen lähdekoodin JavaScript-kirjasto, jonka avulla verkkosovelluskehittäjät voivat luoda koodia tehokkaasti (W3schools 2016).

PHP Personal Home Page on ohjelmistokieli, joka on erityisesti suunnattu palvelinpuolen www-dokumenttien sisältämän ohjelmakoodin suorittamiseen (Rantala 2005, 7).

XAMPP (X)Cross-Platflorm Apache Mysql PHP Pearl on avoimeen lähdekoodiin perustuva www-palvelinohjelmisto joka sisältää tietokantaympäristön sekä tuen HTML-, PHP- ja Pearl-ohjelmointikielille (Apachefriends 2016).

MVC Model View Controller on arkkitehtuuri, jota käytetään yleisesti verkkosovelluskehyksissä, sen avulla mallinnetaan ohjelman rakenne sekä jaetaan sovelluksen toiminnallinen koodi erilleen ulkoasusta (Apple 2015).

1 Johdanto

Opinnäytetyön aiheena oli löytää toimeksiantajan arkistointitietojärjestelmälle uusi toteutusmenetelmä. Opinnäytetyössä tarkasteltavana oleva tietojärjestelmä on vielä prototyyppiversio ja se on toteutettu aiemmin opiskeluun liittyvän työharjoittelujakson aikana. Tämän opinnäytetyön tarkoituksena on kartoittaa nykyisen arkistointitietojärjestelmän puutteita sekä löytää siihen uusi toteutusmenetelmä joka vastaisi nykyistä paremmin modernin verkkosovelluskehityksen kriteereitä. Opinnäytetyön lähtökohdiksi asetettiin myös seuraavia tavoitteita: Oman oppimisprosessini näkökulmasta työn tavoitteena oli perehtyä verkkosovelluskehityksen teknologioihin ja työkaluihin sekä syventää henkilökohtaista osaamista verkkosovelluskehityksen saralla. Toimeksiantajan toiveena oli, kuinka voisin kehittää nykyistä tietojärjestelmää ja toteuttaa sen mahdollisesti toisella tavalla.

Opinnäytetyön lähteinä on käytetty pääasiallisesti painettua kirjallisuutta, koska tietokirjoista löytyy yleensä varmennettua tietoa. Valitettavasti kuitenkin tietokirjat vanhentuvat varsin usein nopean teknologian kehityksen myötä, tämän vuoksi opinnäytetyön lähteiksi on valikoitu uudempia alan tietokirjoja, joissa tieto on ajankohtaista. Opinnäytetyöstä saatujen tulosten perusteella kootaan ja analysoidaan nykyisen tietojärjestelmän kehityskohteet. Samalla pyritään löytämään uusi toteutustapa tai verkkosovelluskehys, jolla nykyinen tietojärjestelmä voidaan korvata.

2 Verkkosovelluskehityksen teknologiat ja evoluutio

2.1 World Wide Web

Nykyisen verkkosovelluksien kehityshistoria voitaneen aloittaa siitä hetkestä, kun Tim Berners-Lee julkaisi World Wide Webin vuonna 1991. World Wide Web kehitettiin alkujaan CERN-tiedeyhteisön tutkijoita varten ja sen oli tarkoitus toimia julkaisualustana sekä arkistointitietojärjestelmänä tiedeyhteisön sisällä tieteellisiä julkaisuja varten. World Wide Webin ideana oli, että CERN:in tutkijat voisivat julkaista ja muokata tutkimuksiaan tai linkittää ne eteenpäin toisten tutkijoiden luettavaksi. World Wide Webin sisältöä varten luotiin HTML-dokumentit, joiden avulla tutkijat pystyisivät lisäämään sisältöä julkaisualustaan (W3C 2016).

Tim Berners-Lee kehitti samaan aikaan maailman ensimmäisen internetselaimen NexTStep-käyttöjärjestelmään. Se oli nimeltään World Wide Web browser / editor ja sen avulla HTML-dokumentteja pystyi lukemaan sekä editoimaan. Hieman myöhemmin, kun internetselaimet kehittyivät ja yleistyivät muihin käyttöjärjestelmäalustoille, selaimissa pystyttiin esittämään muutakin sisältöä kuin pelkkää tekstisisältöä (W3C 2016).

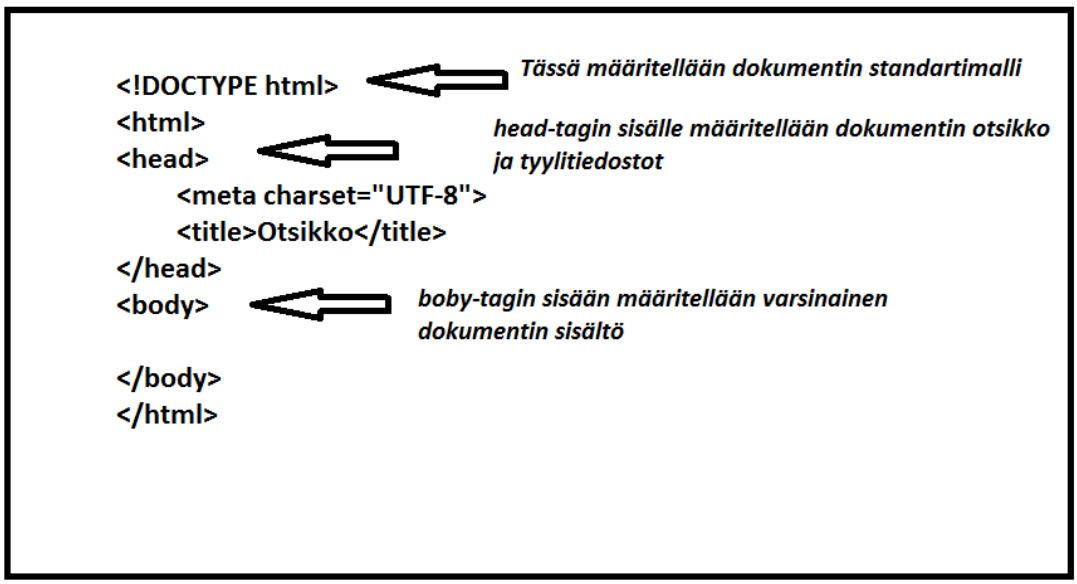
Pääsääntöisesti HTML-dokumenttien sisältö oli kuitenkin staattista, ja niitä päivitettiin vain tarpeen vaatiessa. Smith ja Negrino (2007,1) mainitsevat kirjassaan, että Internetin yleistyessä HTML-dokumenttien sisällön tuottajat halusivat saada entistä enemmän valtaa siihen miltä dokumentit näyttäisivät. Tämä puolestaan edisti HTML-kielen kehittymistä monimutkaisemmaksi.

Sisällön dynaamisuutta varten kehitettiin mm. JavaScript-ohjelmointikieli. Sen kautta selaimessa voitiin suorittaa HTML-dokumentin sisällä reaaliaikaisesti erilaisia toiminnallisia funktioita. Dokumenttien ulkoasua varten kehitettiin CSS-tyylikirjasto, jonka avulla HTML-dokumentin graafista näkymää voitiin hallita dynaamisesti. (Smith & Negrino 2007,1).

2.2 HTML5

HTML on verkkosivuilla käytetty merkkauskieli, sen avulla tietokoneen selaimet osaavat lukea sekä tulkita internetissä olevaa verkkosivuston sisältöä ja rakennetta. Merkkauskielessä käytetään selkokielistä tekstiä, jotta sen tulkkaaminen olisi myös ihmisille ymmärrettävässä muodossa. HTML-dokumentin rakenne koostuu kolmesta osioista, joita ovat dokumentin ***standardimalli, head*** **ja** ***body-***osio. (Korpela 2005, 70).

HTML-dokumentin alussa määritellään käytettävä standardi, head-osioon määritellään dokumentin kuvaava otsikko ja verkkosivuston alussa ladattavat tyyli sekä määritysasetukset. **Body**-osiossa esitetään varsinainen dokumentin sisältö (Kuva 1).



Kuva 1. Esimerkki HTML5:n struktuurista.

Korpela (2011, 24) mainitsee, että 1990-luvun alkupuolella HTML:stä ei ollut käytössä yhtenäistä standardia, vaan sen kehitys oli määrittynyt lähinnä eri selainten ominaisuuksien mukaan. Vuodesta 1995 lähtien merkkauskieltä sekä dokumentin rakennetta lähdettiin määrittelemään yhdenmukaiseksi, ja sitä varten luotiin HTML 2.0-standardi.

Vuonna 1997 HTML-standardia päivitettiin jopa kaksi kertaa, ensiksi tammikuussa versioon 3.2 ja myöhemmin joulukuussa 4.0- versioon. Suurin muutos oli tuki tuleville ohjelmointikielille sekä CSS-tyylikirjastoille, se mahdollisti taas dokumentin ulkoasun määritysten tallennuksen erilliseen tiedostoon. (Korpela 2011, 24).

HTML5:n kehitys aloitettiin kuitenkin vasta 2004. Silloin Apple, Mozilla ja Opera perustivat WHATWG-yhteisön. Yhteisö lähti kehittämään uutta HTML-versiota, joka perustuisi vanhojen standardien pohjalle. Vuodesta 2007 lähtien HTML5:tä on kehitetty yhteistyössä W3C:n kanssa ja samalla sen suosio kasvanut verkkosovelluskehittäjien keskuudessa. HTML5 on periaatteessa evoluution tulos eikä revoluution, sillä se tukee myös vanhaa HTML-kielen rakennetta. (Korpela 2011, 24).

HTML5:n kehitys vanhan pohjalle ei kuitenkaan ollut täysin itsestäänselvyys, sillä aiemman HTML-dokumentin rakenteet ja standartit olivat osaltaan vanhentuneita tai puutteellisia. Jossain vaiheessa yhteisö mietti jopa kirjoittavansa HTML-dokumentin rakenteen uudestaan, tällöin HTML5:ssä ei olisi ollut ollenkaan vanhan rakenteen tuomaa painolastia. (Korpela 2011, 24).

HTML5 tuo uusia ominaisuuksia erityisesti HTML-dokumentin interaktiivisiin toimintoihin, joita ovat mm. piirtoalusta ja tunnettujen video-formaattien tukemien ilman lisäosia. Uutena ominaisuutena on myös paikkatietopalvelu, sen avulla dokumentin sisältö voidaan suoraan toteuttaa sijaintitiedon perusteella.

2.3 JavaScript

Smith ja Negrino (2007,5) määrittelevät JavaScriptin seuraavasti: JavaScript on ohjelmointikieli, jonka avulla HTML-dokumenttiin voi rakentaa dynaamista interaktiivisuutta. JavaScriptin kehitti Brendan Eich, joka työskenteli Netscape Communications-yrityksessä.

JavaScript julkaistiin vuonna 1995. JavaScript tunnettiin aluksi työnimellä Mocha, myöhemmin sen nimi muutettiin muotoon LiveScript. Samaan aikaan kun LiveScript julkaistiin, Java-ohjelmointikielestä povattiin seuraavaa uutta ohjelmointikieltä tietojenkäsittelyssä. Tämän vuoksi Netscapen markkinointi-osasto päätti nimetä LiveScriptin suoraan JavaScriptiksi. Netscape toivoi, että nimen vaihdoksen jälkeen Javan suosio tarttuisi myös heidän kehittämäänsä ohjelmointikieleen. (Smith ja Negrino 2007,5).

JavaScript on asiakaspuolen ohjelmointikieli, ja se suorittaa ainoastaan ohjelmakoodin käyttäjän selaimessa. Koska ohjelmakoodin suoritus tapahtuu selaimessa, JavaScriptiin on lisätty muutamia rajoituksia turvallisuussyistä. Smith ja Negrino (2007,7) mainitsevat rajoitteet seuraavasti: JavaScriptissä on estetty asiakastietokoneen sisältämien tiedostojen lukeminen ja kirjoittaminen. JavaScript ei myöskään salli tiedostojen luomista suoraan palvelimelle, vaan se vaatii erillisen rajapinnan palvelinpuolelle.

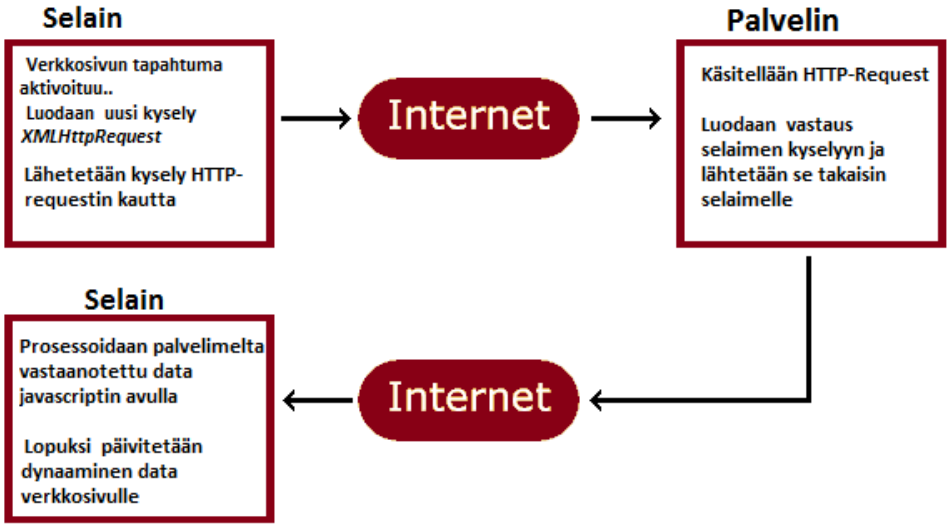
JavaScript ei pysty myöskään sulkemaan ohjelma-ikkunoita, jos niitä ei ole luotu samalla ohjelmointikielellä. JavaScript ei kykene lukemaan tai tutkimaan toisen verkkosivun tietoja, jos ne ovat peräisin toiselta palvelimelta. Käytännössä tämä tarkoittaa sitä, että JavaScript ei voi lukea avoimia sivua ja tutkia esimerkiksi missä käyttäjä on aiemmin käynyt.

JavaScript-ohjelmointikielen avulla voidaan luoda verkkosivustolle interaktiivinen käyttöliittymä. Sillä voidaan dynaamisesti ohjata myös HTML-dokumentin CSS-tyylikirjastoja, näin HTML-dokumenttiin saadaan reaaliaikaisia toimintoja tai visuaalisia tehosteita. Tehosteilla ja CSS-tyyleillä pyritään saamaan verkkosivuston sisältö paremmin näkyviin. JavaScript-ohjelmointikieltä hyödynnetään usein ohjaamaan käyttäjää esimerkiksi erilaisten lomakkeiden täyttämisessä (Włodarczyk 2014).

2.3.1 Ajax

Ajax on JavaScriptissä yleisesti käytettävä viestinvälitysteknologia. Sen avulla voidaan tuoda verkkosivustolle dynaamista tietoa palvelimelta tai suoraan toiselta verkkosivulta. Ajaxin avulla voidaan siis hakea palvelimelta reaaliaikaista dataa ilman, että koko verkkosivustoa tarvitsee erikseen päivittää uudestaan selaimessa. Ajax koostuu pääsääntöisesti kolmesta objektista, joita ovat XMLHttpRequest ja XMLHttpResponse sekä onreadystatechange-tapahtumametodi (W3schools 2016).

XMLHttpRequest lähettää selaimen kyselyn palvelimelle. ***XMLHttpResponse*** käsittelee palvelimelta saadun datan ja tulostaa sen esimerkiksi tietyille **DOM-elementeille**. **Onreadystatechange**-tapahtumametodi tarkastaa selaimelta lähetetyn kyselyn statuksen palvelimelta. Ajax käsittelee ja tulostaa palvelimelta saadun datan teksti- tai XML-muotoisena (Kuva 2).

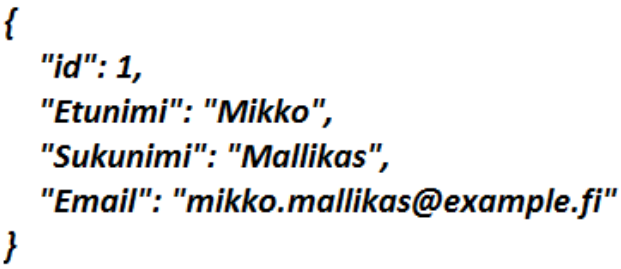


Kuva 2. Toimintakaavio kuinka Ajax-käsittelee suoritukset.

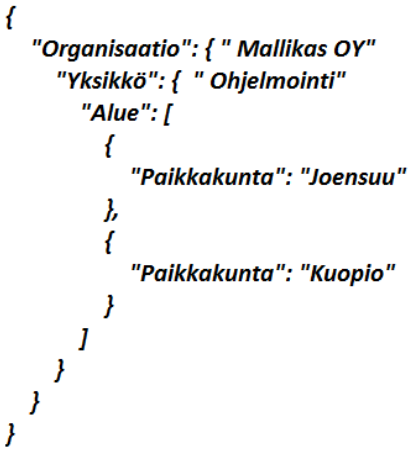
2.3.2 JSON

JSON on verkkosovelluksissa käytettävä hierarkkinen tiedonvälitys-standardi. Sen avulla käsiteltävää dataa voidaan siirtää tietojärjestelmästä toiseen varsin nopeasti ja luettavassa muodossa. Vaikka JSON on lyhennys sanasta JavaScript Object Notation, se tukee myös muita ohjelmointikieliä, joten sitä voidaan käyttää universaalisti (W3schools 2016).

JSON on suunniteltu vaihtoehdoksi XML-standardille ja se sopii erityisen hyvin NO SQL-pohjaisten tietokantojen tiedonvälitykseen. JSON-dataformaatti rakentuu aaltosulkeiden sisään ja aaltosulkeiden sisällä on arvoparit eli muuttujan nimi ja muuttujan arvo (Kuva 3). Data voidaan luoda myös sisäkkäisessä muodossa (Kuva 4).



Kuva 3. Esimerkki yksinkertaisesta JSON-datarakenteesta.



Kuva 4. Esimerkki sisäkkäisestä JSON datarakenteesta.

2.3.3 jQuery

jQuery on JavaScript-ohjelmointikielelle tehty sovellus. Sen avulla JavaScript koodia voidaan suorittaa sekä kirjoittaa tehokkaammin. Lisäksi jQuery tukee Ajaxia sekä siihen voidata ladata erilaisia kolmannes osapuolen laatimia lisäosia. jQuery otetaan käyttöön verkkosovelluksessa lisäämällä viittaus tai linkki jQueryn kirjastotiedostoon. Kirjastotiedosto voidaan määritellä ja asentaa paikallisesti verkkosovelluksen palvelimelle tai sitä voidaan käyttää etänä jQueryn verkkosivuston kautta (W3schools,2016).

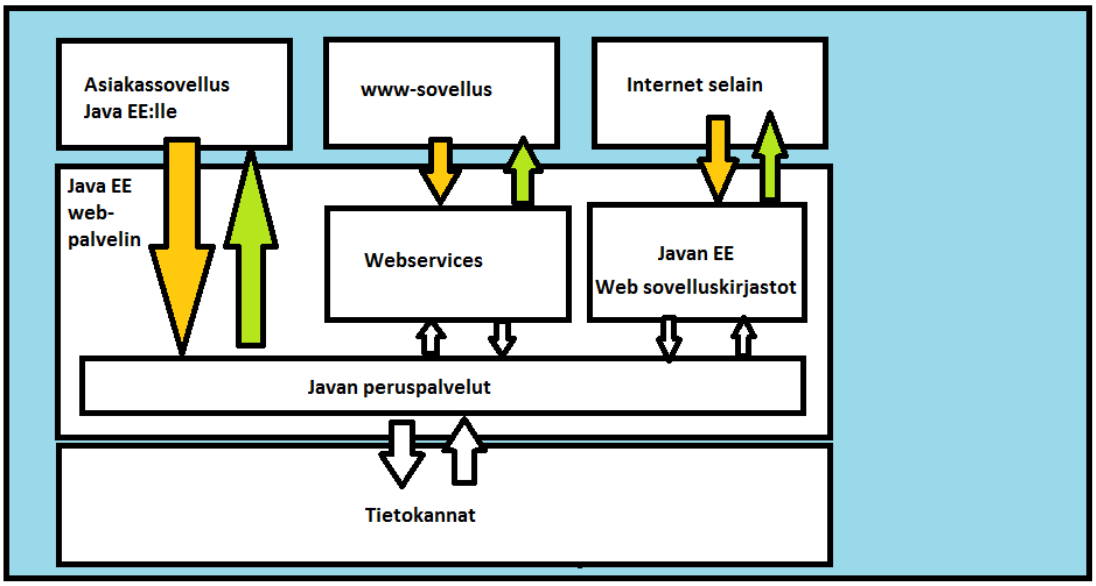
jQuery on varsin voimakas työkalu, sillä sen avulla verkkosovelluskehittäjä voi kirjoittaa JavaScript koodia lyhyemmillä komennoilla, näin sovelluksen koodi saadaan myös huomattavasti lyhyemmäksi. jQueryn kotisivuilla löytyy laaja dokumentaatio kirjaston käytöstä sekä tuetuista komennoista, ja dokumentaatiota päivitetään varsin usein, etenkin silloin kun siihen tulee uusia ominaisuuksia tai sen versio päivittyy uudempaan.

2.4 Java EE

Westerholm ja Kyyppö (2015,19) mainitsevat, että Java kehitettiin alun perin ohjelmointikieleksi sulatettuja järjestelmiä sekä pienelektroniikkaa varten. Javan kehitti Sun Microsystemsin työntekijä James Gosling vuonna 1991, ja se tunnettiin aluksi nimellä Oak. Java-ohjelmointikielen ajatuksena oli poistaa ohjelmien uudelleen kääntäminen eri prosessori-arkkitehtuureille ja näin luoda ohjelmointikielestä alustariippumaton.

World Wide Webin yleistyessä Oak-kieltä lähdettiin kehittämään myös internetissä käytettäväksi. Sun Microsystem julkaisi vuonna 1994 HotJava-internetselaimen, ja hieman myöhemmin vuonna 1995 yhtiö päätti, että Oak-nimi muutetaan Javaksi. (Westerholm & Kyyppö 2015,19).

Vuonna 1999 julkaistiin Java2EE, joka suunniteltiin yhdistämään verkkosovelluskehitys ja Java ohjelmointikieli. Myöhemmin nimi muuttui Java EE:ksi. Java EE on käytännössä kokoelma erilaisia verkkosovellusohjelmia tai rajapintoja. Java EE on palvelinohjelmisto, joka tarjoaa verkkosovelluskehittäjille laajan valikoiman valmiita Java-kirjastotiedostoja sekä mahdollisuuden hyödyntää myös muita verkkosovellusteknologioita (Kuva 5) (Westerholm & Kyyppö 2015,590).



Kuva 5. Java EE:n toiminnallinen rakenne (Westerholm & Kyyppö 2015,590).

2.4.1 Maven

Maven on Apache Software Foundation kehittämä työkalu verkkosovelluskehitykseen, se on erityisesti suunniteltu helpottamaan Java EE-sovellusten rakentamista. Mavenin avulla käytännössä määritellään rakennettavan sovelluksen projektinhallinta. Kuha (2008,45) mainitsee, että Mavenin avulla projektiin pitäisi määritellä vähintään seuraavat asiat: kuinka verkkosovellus pitäisi rakentaa, mitä osioita siihen liittyy sekä kuinka se toimitetaan ja paketoidaan tuotantokäyttöön tai suoraan loppukäyttäjälle.

Maven löytyy valmiiksi asennettuna tai se on asennettavissa plugin-lisäosana Java IDE-kehitysympäristöihin, näin verkkosovelluskehittäjän ei välttämättä tarvitse asentaa sitä erikseen Apache Software Foundationin kotisivuilta. Maven projektimalli otetaan samalla käyttöön, kun lähdetään luomaan uutta Java EE-ohjelmointiprojektia. Maven laatii automaattisesti verkkosovellukseen kansiohierarkian joka noudattaa seuraavaa rakennetta. (Kuha 2008,53) (Taulukko 1).

Taulukko 1. Mavenin hakemistorakenne

src/main/java	Sisältää projektin lähdekoodit ja tiedostot
src/main/resources	Sisältää projektin resurssitiedostot (sisältää xml-tiedostot)
src/main/filters	Sisältää tiedostot ympäristön riippuvuuksia varten
src/main/assembly	Sisältää lähdekoodin koostamiseen tarvittavat tiedostot
src/main/config	Sisältää projektiin liittyvät asetus ja määritystiedostot
src/main/webapp	Java EE verkkosovelluksen resurssit
src/test/java	Sisältää testauksen lähdekoodit
Src site	Projektin informaatiosivu

Mavenin kautta voidaan määritellä kehitysympäristö vastaamaan myös täysin tuotantoympäristöä. Silloin pystytään eliminoimaan mahdollisia virhetilanteita, kun projektia lähdetään siirtämään tai ottamaan käyttöön varsinaisessa tuotantoympäristössä. Maven perustuu projektimalliin, joka tunnetaan yleisesti nimellä Project Object Model, eli lyhennettynä POM. (Kuha 2008,45).

Ohjelmistoprojektin rakenne kuvataan kokonaisuudessaan yhdessä Pom.xml-tiedostossa (Kuva 6). XML-tiedostoon kirjataan ohjelmakoodin kääntämiseen, testaukseen sekä raportointiin liittyvät seikat (Taulukko 2). Projekti voidaan suoraan testata, kääntää tai paketoida Mavenin avulla.

Mavenin komennot suoritetaan komentorivin kautta mvn-komennolla tai Javan IDE-kehitysympäristö huolehtii automaattisesti Mavenin komentojen toteutumi-

sesta. Samankaltaisissa projekteissa voi toistua tietyt perusasiat, silloin voidaan hyödyntää Mavenin omia ns. arkkityyppejä. Ne ovat käytännössä valmiita projektipohjia. Arkityyppien avulla uusi ohjelmointiprojekti voidaan käytännössä aloittaa suoraan eikä verkkosovelluskehittäjän tarvitse tehdä projektiin uudestaan samoja alkuvalmisteluja. (Kuha 2008,45).

.

```
<project>
   <parent>
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.IT-ykkoset.app</groupId>
        <artifactId>Hotellivaraus-app</artifactId>
        <version>1</version>
       <packaging>jar</packaging>
  </parent>
        <modelVersion>4.0.0</modelVersion>
        <artifactId> Asikkaan sisäänkirjaus</artifactId>
        <version>2.0</version>
        <packaging>jar</packaging>
</project>
```

Kuva 6. Esimerkki POM.xml tiedostosta.

Taulukko 3. Kuvaus POM.xml- tiedoston rakenteesta

<ModelVersion>	Projektimallin versio
<groupId>	Sovelluksen ryhmätunniste
<version>	Versionumero projektista
<arftifactID>	Yksilöllinen ryhmätunniste, joka on tarkoitettu lähinnä projektin sisäiseen käyttöön
<parent>	Viittaus sidoksissa olevaan ylempään projektiin. Käytetään projekteissa, jotka sisältävät alimoduuleita.
<packaging>	Pakkausmuoto missä projekti toimitetaan asiakkaalle. Esim. jar-tiedostona
<name>	Kuvaava projektin nimi
<description>	Lyhyt kuvaus projektista ja sen tarkoituksesta.

2.4.2 Apache Ant

Apache Ant on Apache Software Foundation kehittämä työkalu, joka on suunniteltu Java EE-verkkosovellusten rakentamiseen ja kääntämiseen. Apache Ant sisältää samat toiminnot kuin Maven, mutta sen toimintalogiikka eroaa huomattavasti Mavenista, sillä sen toiminta perustuu komentoskripteihin, joiden avulla projektin luomista voidaan automatisoida. (Kuha 2008,371).

Se asennetaan Mavenin tavoin lisäosana kehitysympäristöön. Projektin asetukset määritellään build.xml-Tiedostoon joka sijaitsee yleensä projektin juurikansiossa. Apache Ant ei luo automaattisesti Java EE-projektille kansiorakennetta, vaan verkkosovelluskehittäjän on määriteltävä projektin kansiorakenne itsenäisesti build.xml tiedostoon.

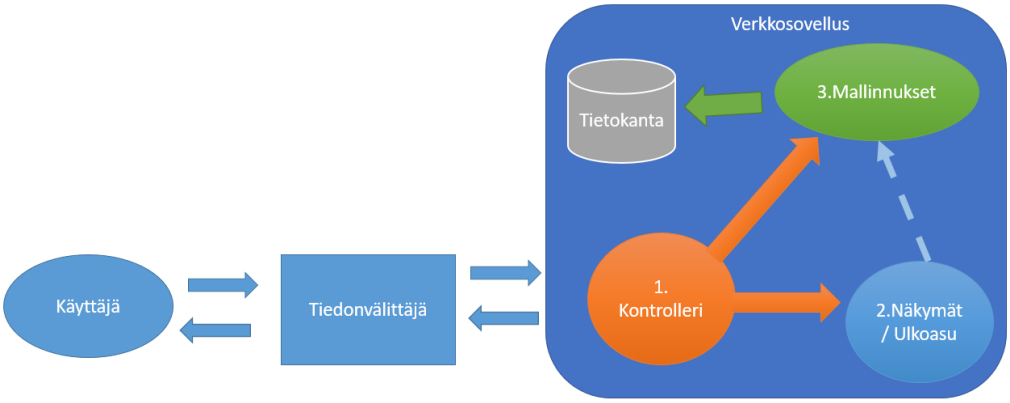
Build.xml tiedoston määrittely aloitetaan syöttämällä ***<Project name>***-liite, johon määritellään projektin nimi sekä projektin kotikansio. Sen sisällä määritellään projektiin liittyvät suoritettavat toiminnot ja riippuvuudet. Projektin varsinaiseen kääntämiseen, luomiseen sekä poistoihin liittyvät toiminnot määritellään erikseen ***<Target>*** -liitteiden sisään (Taulukko3). Lopuksi kun xml-tiedosto on määritelty, niin se ajetaan Java-kehitysympäristössä tai komentoriviltä.

Taulukko 3. Esimerkki build.xml tiedostosta ja toiminnallinen kuvaus.

<project name="projekti" default="deploy" basedir=".">	1. Tässä määritellään projektin nimi ja juuri-kansion sijainti
<target name="puhdista"> *<deltree dir="${outputDir}" />* *</target>*	2. Tässä target-komennolla puhdiste-taan projektin kansio. Tagin sisälle määritelty deltree-poistokomento.
<target name="valmistele" *depends="puhdista">* *<mkdir dir="${outputDir}" />* *</target>*	3. Tässä luodaan uusi kansio. Tagin sisään määritelty mkdir-komento joka luo uuden kansion. Komento sisältää riip-puvuuden **”puhdista”** tagiin, joten sitä ei suo-riteta ennen kuin ”**puhdista”** on suoritet-tu.
<target name="käännä" depends="valmistele"> *<javac srcdir="${sourceDir}"* *destdir="${outputDir}" /> </target></project>*	4. Lopuksi projekti käännetään uuteen kansioon. Tämä teh-dään vasta kun ”val-mistele” -tagi on suori-tettu.

2.4.3 MVC-arkkitehtuuri

MVC-arkkitehtuuria käytetään useissa verkkosovelluskehyksissä, sillä sen avulla verkkosovelluksesta voidaan tehdä entistä modulaarisempi ja sen kautta verkkosovelluksen moduuleita on myös huomattavasti yksinkertaisempi ylläpitää. Käytännössä MVC-arkkitehtuuri tarkoittaa sitä, että ohjelman toiminnallinen koodi pidetään erillään verkkosovelluksen HTML-dokumentin ulkoasusta, samalla verkkosovelluksen (Kuva 7) rakennetta voidaan mallintaa paremmin käyttötarkoitusta varten (Apple 2015).



Kuva 7. MVC-arkkitehtuurin toimintarakenne

Kontrolleriin määritellään verkkosovelluksen toiminnallisuus ja se vastaa käytännössä verkkosovelluksen sisäisten rajapintojen tiedonvälityksestä. Näkymissä määritellään verkkosovelluksen ulkoasuun sekä visuaaliseen ilmeeseen liittyvät asetukset ja muutokset. Mallinnuksissa huolehditaan siitä, että tietokannan rakenne vastaa sovelluksen käyttötarkoitusta.

2.4.4 Hibernate

Hibernate on tietokannan käsittelyyn erikoistunut olio-relaatiosovelluskehys. Sen avulla Java-alustaiset verkkosovellukset pystyvät kommunikoimaan avulla erilaisten tietokantajärjestelmien kanssa olio-arkkitehtuurin mukaisesti (Kuha 2008,374). Hibernate ei ole automaattisesti Java IDE-kehitysympäristöiden mukana vaan se on erikseen asennettava kehitysympäristöön. Asennuksen jälkeen hibernate on käytettävissä uusiin verkkosovellusprojekteihin.

Hibernate otetaan käyttöön uuteen projektiin yleisesti seuraavalla tavalla: Ensiksi luodaan kaksi XML-tiedostoa hibernate.cfg.xml ja hibernate.mapping.xml sekä liitettävät luokat jotka toimivat verkkosovelluksen apuna, kun tietokannan tietoja käsitellään. Cfg.xml-tiedostoon määritellään tietokantojen yhteysasetukset käytettävät ominaisuudet sekä viittaus mapping.xml-tiedostoon. Mapping.xml-tiedostoon määritellään oikeastaan vain tietokannan käsittelyyn liittyvän luokan jäsenmuuttujat sekä niiden tietotyypit, jotta hibernate osaa hyödyntää ko. luokan aksessoreita ja mutaattoreita.

Tämän jälkeen hibernate.jar tiedosto lisätään vielä projektin kirjastokansioon. Nyt hibernate rajapintaa voi hyödyntää myös varsinaisessa ohjelmakoodissa. Hibernaten etuna on se, että tietokannan asetuksia ja parametreihin liittyviä asetuksia tarvitsee muuttaa vain hibernate xml-tiedostoihin eikä ohjelmakoodiin itsessään.

3 Nykyisen tietojärjestelmän kehityskohteet

3.1 Tietojärjestelmän nykytila ja kehityshistoria

Tietojärjestelmän nykyinen tilanne on se, että toimeksiantajalla on toimiva prototyyppiversio käytössään, mutta sen interaktiivisiin toimintoihin kaivataan erityisesti reaaliaikaisuutta sekä käyttöliittymään selkeää visuaalista parannusta. Ylläpidon ja tietoturvan näkökulmasta nykyinen tietojärjestelmä ei myöskään täytä täysin sille asetettuja kriteereitä.

Nykyinen tietojärjestelmä toimii työasemassa, johon on asennettu XAMPP-palvelinohjelmisto. Sovellus toimii siten, että käyttäjä kirjautuu saaduilla tunnuksilla tietojärjestelmään ja kirjautumisen jälkeen käyttäjä voi esimerkiksi arkistoida asiakirjoja ja katsella niitä myöhemmin. Tietojärjestelmä on toteutettu pääosin PHP sekä HTML-ohjelmointikieltä käyttäen. Tietojärjestelmässä on myös hyödynnetty hieman JavaScript-ohjelmointikieltä ja käyttöliittymän osalta Bootstrap-verkkosovelluskehystä. Sovelluksen tietokantaympäristönä toimii MySQL-tietokantajärjestelmä.

Työharjoittelun aikana tietojärjestelmästä luotiin vaatimusmäärittely ja sen perusteella kartoitettiin tietojärjestelmälle alustava käyttötarkoitus ja toimintaidea. Vaatimusmäärittelyssä ja suunnittelussa kesti useampi viikko, jotta saatiin kokonaiskuva arkistointisovelluksen toimintaperiaatteesta. Vaatimusmäärittelyn jälkeen lähdettiin ohjelmoimaan tietojärjestelmän nykyistä prototyyppiä.

3.2 Tietojärjestelmän ylläpito

Tietojärjestelmän vaivaton ylläpito on nykypäivänä tärkeä asia. Oleellista on siis se kuinka nopeasti ja mutkattomasti verkkosovellusta voidaan päivittää tai tarpeen vaatiessa muokata. Nykyisen tietojärjestelmän yksi ongelmakohta on se, että se on vaikea ylläpitää, ja sovelluksen koodin modulaarisuutta on hyödynnetty varsin vähän.

Käytännössä tämä tarkoittaa sitä, että jos sovellukseen tulee muutostarvetta, niin muutoksia joutuu tekemään useampaan kuin yhteen kohtaan sovelluksen koodissa. Tämän vuoksi mietin valmiiden sovelluskehysten käyttöä osana arkistointisovellusta, koska sovelluskehysten avulla saadaan pilkottua koodi pienempiin moduuleihin. Samalla sen ylläpito- ja päivitystoiminnot helpottuvat myös huomattavasti sekä sovelluksen toiminnallinen koodi pystytään erittelemään sovelluksen ulkoasusta tai varsinaisesta HTML-koodista.

Sovelluskehykset hyödyntävät myös koodin uudelleen käyttöä olio-orientoineesti. Olio-orientoituneesta ohjelmoinnista saadaan myös merkittävää hyötyä, kun sovelluksen tietokantaa tarvitsee käsitellä. Ohjelmoijan näkövinkkelistä sovelluskehykset tarjoavat ketterän keinon ylläpitää sovelluksia, ja samalla sovelluskehykset huolehtivat siitä, että sovelluksen yhteydessä käytettävät kirjastot sekä niiden riippuvuudet ovat aina ajan tasalla.

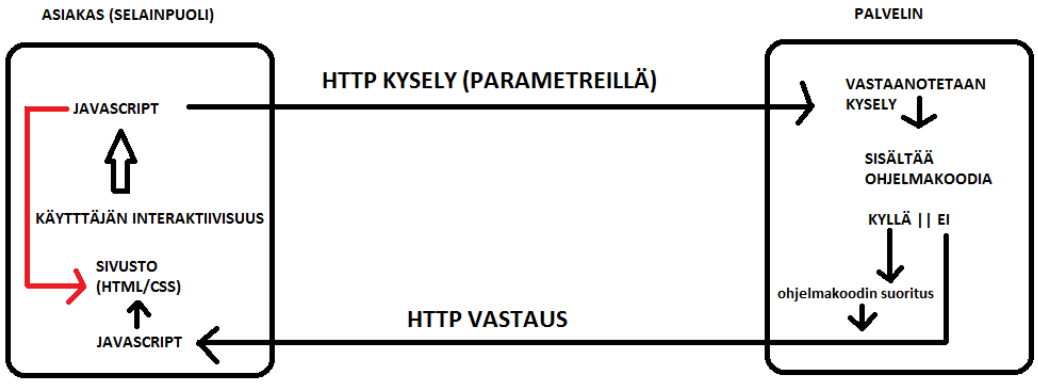
Käyttäessä sovelluskehyksiä lisäarvoa saadaan myös siitä, että koodia ei tarvitse kirjoittaa yhtä paljon kuin normaalisti, koska useat sovelluskehykset osaavat myös hyödyntää kolmannen osapuolen tekemiä sovelluksia esimerkiksi (JQuery), joiden avulla voidaan kirjoittaa monimutkikkaita toimintoja lyhyesti sekä ytimekkäästi.

Laitteistokannan puolesta ylläpito on järkevää siirtää oikealle fyysiselle palvelimelle tai suoraan valmiiseen pilvipalveluun joka tukee tarvittavia sovelluskehyksiä. Näin vikatilanteen sattuessa sovelluksen varmuuskopiointi on riittävän hyvin varmistettu Raid-levyjärjestelmällä sekä erillisillä varmuuskopioilla.

3.3 Tietojärjestelmän interaktiivisuus

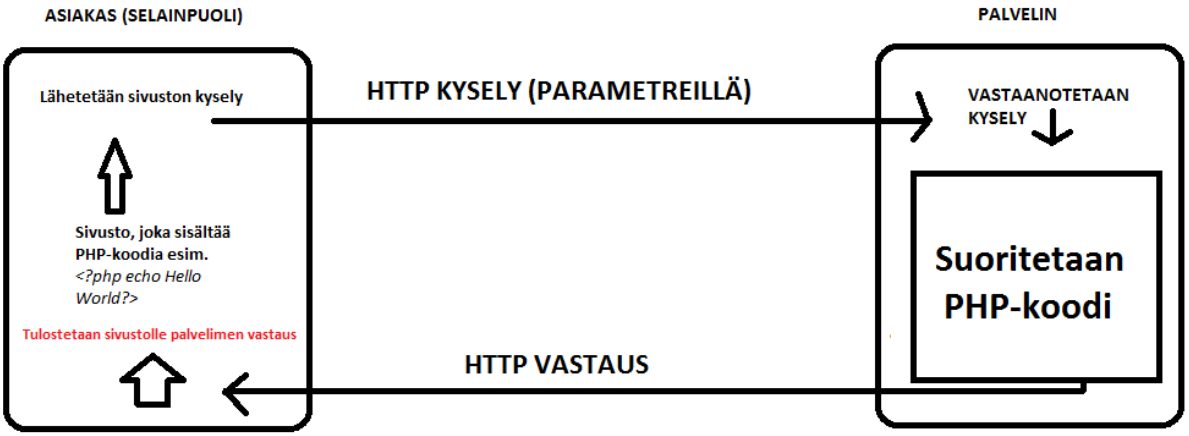
Nykyisen tietojärjestelmän reaaliaikaiset toiminnallisuudet ovat varsin rajalliset, johtuen PHP-koodin suoritusarkkitehtuurista. Sen vuoksi verkkosovelluksen interaktiivisuuden lisääminen ja hyödyntäminen erityisesti selaimen puolella olisi kehittämisen arvoinen asia.

Nykyiset verkkosovellukset pystyvät suorittamaan toiminnallisuuksia ilman merkittäviä viiveitä. Käytännössä tämä tarkoittaa sitä, että verkkosovelluksen käyttäjän ei siis tarvitse lähettää jokaista sovelluksen ohjelmakoodia suoritettavaksi palvelimelle, vaan osa ohjelmakoodista voidaan suorittaa suoraan käyttäjän internetselaimen sisällä reaaliajassa (Kuva 8).



Kuva 8.Selaimen puolella tapahtuvasta skriptauksesta eli ohjelmistokoodin suorituksesta (Alicea 2015).

Kuten aiemmin mainitsin, nykyinen tietojärjestelmä on kirjoitettu PHP-ohjelmointikielellä. Se tukee huonosti tai ei ollenkaan reaaliaikaisia toimintoja, koska ä kun selain suorittaa ohjelmakoodin, se lähettää datan aina suoraan palvelimelle (Kuva 9).



Kuva 9. Esimerkki palvelinpuolella tapahtuvasta ohjelmistokoodin suorituksesta (Rantala 2004, 4).

Vastaus palvelimelta saattaa kuitenkin tulla viiveellä, mikä hidastaa verkkosovelluksen käyttöä ja aiheuttaa turhaa tietoliikennettä koneen sekä palvelimen välille. Ohjelmointiprojektin alussa oli toimeksiantajan kanssa keskustelua, että verkkosovelluksen pitäisi olla mahdollisimman yksinkertainen sekä helppokäyttöinen, jotta käyttäjillä ei olisi suuri kynnys lähteä opettelemaan uutta tietojärjestelmää.

Interaktiivisuuden avulla verkkosovelluksesta voidaan rakentaa entistä suoraviivaisempi ja monipuolisempi sekä käyttäjän näkökulmasta katsoen helppokäyttöisempi. Samalla pystytään vähentämään käyttäjän mahdollisia virhetilanteita, koska käyttäjää voidaan ohjeistaa reaaliajassa sovelluksen käytössä. Reaaliaikainen interaktiivisuus voidaan toteuttaa etenkin JavaScript-ohjelmointikielellä varsin helposti.

3.4 Tietoturva

Nykyiseen tietojärjestelmään on lisätty käyttäjäkirjautuminen sekä järjestelmään kirjautumista rajattu palomuuriin asetuilla säännöillä tietyistä verkoista. Näin voidaan osaltaan estää sovelluksen luvaton käyttö. Lisäksi kansio, johon arkistomateriaalit tallennetaan, on suojattu ulkopuolisilta palvelimeen asetetulla Apachen .htaccess-suojausmäärityksillä.

Tietojärjestelmän käyttäjät on jaoteltu karkeasti kahteen tasoon eli pääkäyttäjiin ja perustason käyttäjiin. Pääkäyttäjät voivat muokata tai poistaa kaikkia arkiston sisältämiä tietoja ja tiedostoja. Peruskäyttäjä pystyvät käsittelemään, lisäämään, ja poistamaan vain omia arkistomateriaalejaan.

Nykyinen suojaustaso ei kuitenkaan riitä vaan myös varsinaista ohjelmakoodia pitäisi pystyä suojaamaan määrätyiltä osin myös ulkopuolisilta tahoilta etenkin tietokantakyselyiden osalta, jottei niihin kohdistuisi yllättäviä palvelunestohyökkäyksiä. Nykyisen tietojärjestelmän ohjelmakoodi on osittain sirpaleista sekä PHP:n ohjelmakoodin kautta dataa käsitellään ajoittain selväkielisenä. Tämä saattaa altistaa tietojärjestelmän useille vakaville yhtäaikaisille hyökkäyksille.

Parantaakseni sovelluksen tietoturvaa sovelluksen ohjelmakoodi pitäisi saada mahdollisimman modulaariseksi sekä pääosin olio-orientoineeksi. Näin ohjelmakoodin luokkia sekä niiden sisältämiä toiminnallisia funktioita tai jäsenmuuttujia voidaan suojata paremmin näkyvyysmääreillä, kuten mm. avainsanoilla ***Protected*** ja ***Private***. Näillä avainsanoilla estetään ulkopuolelta luokan sisällön väärinkäyttö.
Verkkosovellusten kehitykseen on luotu valmiita sovelluskehyksiä, joissa on ajateltu jo valmiiksi sovellusten tietoturvaa, modulaarisuutta, ja etenkin hyödynnetty olio-orientoinutta ohjelmointia. Sovelluskehyksen mukana tulleita kirjastosovelluksia pidetään myös automaattisesti ajan tasalla, mikä parantaa myös itsessään sovelluksen tietoturvaa.

3.5 Käyttöliittymä

Verkkosovelluksen käyttöliittymä on verkkosovelluksen visuaalisen ilmeen ja käytettävyyden kannalta todella tärkeä sekä olennainen asia. Korpela ja Linjamaa (2005, esipuhe) mainitsevat websivuston suunnitellun pääperiaatteiksi, että sivusto olisi suunniteltava mahdollisimman helppokäyttöiseksi sekä verkkosovelluksen tulisi toimia erilaisissa katselutilanteissa, ja tuoda selkeästi näkyviin sovelluksen sisältö.

Nykyisen tietojärjestelmän käyttöliittymän visuaalinen suunnittelu ja toteutus jäivät todella vähäiseksi, koska ensisijainen tarkoitus oli saada sovelluksen toiminnallinen osuus valmiiksi. Arkistointisovelluksen nykyisessä ulkoasussa on kuitenkin hyödynnetty hieman CSS-tyylikirjastoja sekä käytetty Bootstrap-sovelluskehityksen valmiita tyylimäärittelyitä. Arkistointisovelluksen käyttöliittymä on tarkoitus uusia kokonaan, ja tehdä siitä huomattavasti visuaalisempi, samalla on tarkoitus luoda sovelluksesta responsiivinen.

Responsiivisuus tarkoittaa sitä, että sovellusta voidaan käyttää erikokoisilla pääte tai näyttölaitteilla, ja se skaalautuu aina näytölle sopivaksi. Hyvänä esimerkkinä voisi olla, että mobiilikäyttäjä pystyisi sujuvasti käyttämään sovellusta mobiililaitteelta, jonka näyttö on suhteessa pieni verrattuna tietokoneen näyttöön. Käyttäjän käyttökokemuksen osalta sovelluksen interaktiivisuutta, ja reaaliaikaista ohjautuvuutta on tarkoitus lisätä, mutta unohtamatta noudattaa websivuston suunnittelun perusperiaatteita.

3.6 Laajennettavuus

Laajennettavuus eli tietojärjestelmän skaalaavuus on huomioitava myös, kun uusittua arkistointisovellusta lähdetään kehittämään, ja toteuttamaan. Tietojärjestelmän pitäisi olla myös pienellä vaivalla helposti laajennettavissa tai integroitavissa toisiin tietojärjestelmiin. Samalla tietojärjestelmän pitäisi pystyä lähettämään myös dataa sulavasti universaalilla tavalla esimerkiksi webservices- tai SOAP-verkkopalveluiden välityksellä erilaisiin tietojärjestelmiin.

Laajentuvuuden osalta on tarkoitus hyödyntää valmiita sovelluskehyksiä, koska niiden kautta verkkosovellusta voidaan laajentaa yhtenä kokonaisuutena. Lisäksi ne tukevat jo valmiiksi ajantasaisia kirjastotiedostoja tai lisäohjelmia, joilla tietojärjestelmää voidaan laajentaa todella nopeasti sekä kätevästi.

Tietojärjestelmää ei ole kuitenkaan tarkoitus laajentaa massiiviseksi, koska tällöin järjestelmän prosessit voivat monimutkaistua tai pahimmassa tapauksessa hidastua olennaisesti. Tämän opinnäytetyön myötä tietojärjestelmän laajennusmahdollisuudet pitäisi kuitenkin pystyä hahmottamaan tarkasti, eikä ole tarkoituksenmukaistakaan tehdä tietojärjestelmästä laajaa tai raskaasti käytettävää. Lähinnä tarkoitus on, että laajennettavuuden tarve ja kriteerit on huomioitu sovellusta toteuttaessa, koska se voi tarjota mahdollisuuksia kehittää järjestelmää tulevaisuudessakin.

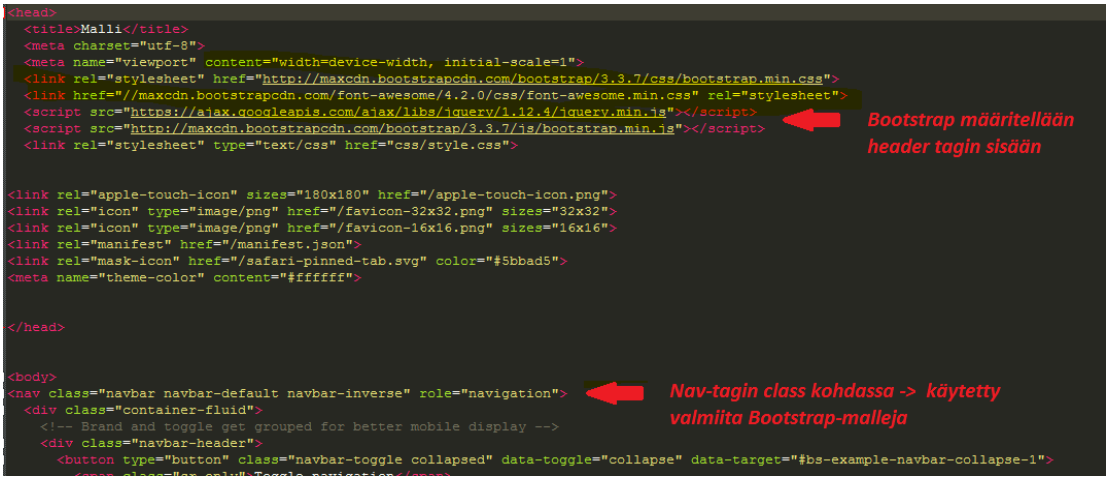
Laajennettavuuden osalta korostuu myös aiemmin mainitsemani tietojärjestelmän ohjelmakoodin modulaarisuus ja olio-orientoisuus. Hyvä verkkosovellus koostuu useista moduuleista, ja verkkosovelluksen moduulit voivat toimia joko itsenäisesti omana yksikkönään tai kommunikoida yhdessä toisen moduulin kanssa. Moduuleita voidaan ajatella esimerkiksi legopalikkoina eli yksittäisinä objekteina. Kun legopaloja kasataan yhteen, niistä voidaan rakentaa mm. lego-

talo, joka on myös itsessään objekti. Kuvainnollisesti legotalo voisi olla arkistointisovellus, ja legopalikka verkkosovelluksen laajennusosa tai moduuli.

4 Sovelluskehykset

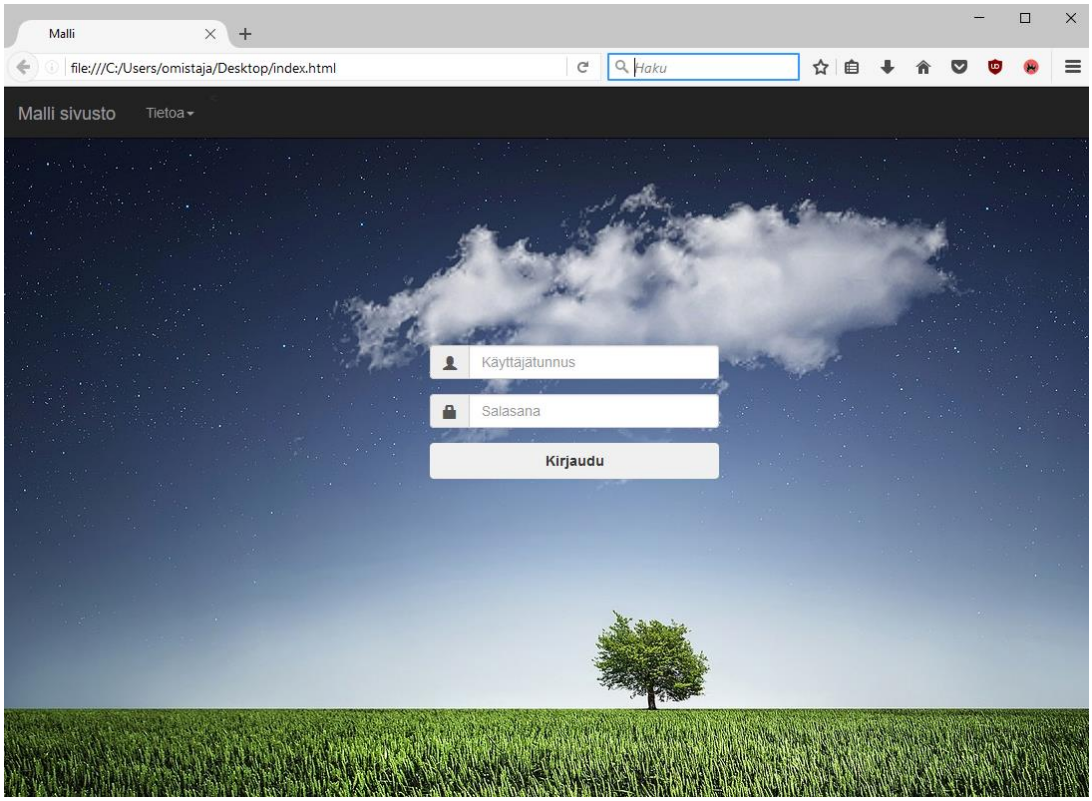
4.1 Bootstrap

Bootstrap on avoimeen lähdekoodiin perustuva verkkosovelluskehys, jonka avulla verkkosovelluskehittäjät voivat luoda nopeasti verkkosivujen käyttöliittymiä sekä määritellä dokumentin ulkoasuun liittyviä seikkoja. Se sisältää valmiita typografioita, painikkeita sekä lomakemalleja, ne on tehty CSS-tyylikirjaston määritysten mukaan. Sen avulla voidaan toteuttaa myös verkkosovelluksen sivustot responsiiviseksi tällöin verkkosivustot toimivat pienemmillä pääte sekä näyttölaitteilla oikein (Kuva 10).



Kuva10. Bootstrap-sovelluskehyksen ominaisuuksia hyödynnetty HTML:ssä.

HTML:n ja CSS:n lisäksi Bootstrap tukee JavaScript-ohjelmointikieltä. Verkkosovelluskehittäjä voi kätevästi myös lisätä oman CSS tai JavaScript ohjelmakoodin Bootstrap-asetuksiin. Bootstrap-verkkosovelluskehys voidaan asentaa fyysisesti paikallisesti samaan paikkaan kuin verkkosivusto tai se voidaan ladata suoraan joka kerta Bootstrapin sivulta.



Kuva11.Esimerkki sovelluskehyksellä toteutetusta visuaalisesta näkymästä.

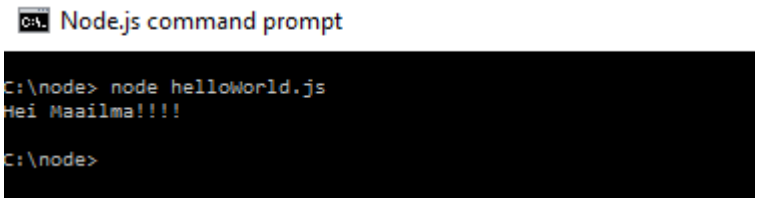
4.2 Node.JS

Node.JS:n on JavaScript-pohjainen verkkosovelluskehys, sen kehitti Ryan Dahl vuonna 2009. Se on kirjoitettu C++-ohjelmointikielellä. Sen taustalla toimii Googlen kehittämä V8-JavaScript-moottori, joka kääntää JavaScript koodin suoraan konekieleksi. Node.js on muutaman vuoden sisällä noussut verkkosovelluskehittäjien suosioon, Sen etuina pidetään erityisesti pitkälle vietyä palvelinpuolen JavaScript tukea, säikeiden asynkronista suoritusta, koodin modulaarisuutta sekä reaaliaikaisten ohjelmien suoritusta. (Alicea 2015).

Node.JS:llä toteutettavassa verkkosovelluksessa tarvitaan vain JavaScript-ohjelmointikieltä ja sen tekniset vaatimukset ovat varsin vaatimattomat. Käytännössä se vaatii vain tietokoneen sekä verkkoyhteyden. Se toimii erinomaisesti Linux-, MacOS- ja Microsoft Windows-alustoilla, joten se voidaan varsin helposti asentaa erilaisiin pilvipalvelimiin tai sulautettuihin järjestelmiin. Node.JS tukee myös laaja-alaisesti erilaisia lisäohjelmia Node Package Managerin kautta. (Percival 2015).

Verkkosovelluskehyksen asennus tapahtuu lataamalla käyttöjärjestelmälle sopiva asennuspaketti. Nodesta pystyy lataamaan kaksi eri versiota. Ensimmäinen on LTS-versio, eli vakaampi pitkän tuen omaava versio. Toinen on taas viimeisin ja uusin versio, joka sisältää uusimmat ominaisuudet. Asennuspaketin suorittamisen jälkeen Node.js on täysin käyttövalmis.

Node.js käynnistetään komentorivillä yksinkertaisesti komennolla **node**. JavaScript koodia voidaan suorittaa Noden kautta esimerkiksi kirjoittamalla komentoriville node helloWorld.js (Kuva12). Node.js-verkkosovelluskehys ei sisällä oletuksena http-palvelinta, mutta sen saa varsin helposti käyttöön kirjoittamalla muutaman rivin JavaScript koodia (Kuva13).



Kuva12. Node.js komentotulkki suorittamassa JavaScript koodia.

```
1   var express = require('express')
2   var app = express()
3
4   app.get('/', function (req, res) {
5     res.send('Hei maailma!!!')
6   })
7
8   app.listen(3000, function () {
9     console.log('Käynnistetään web-palvelin: Portti 3000')
10  })
```

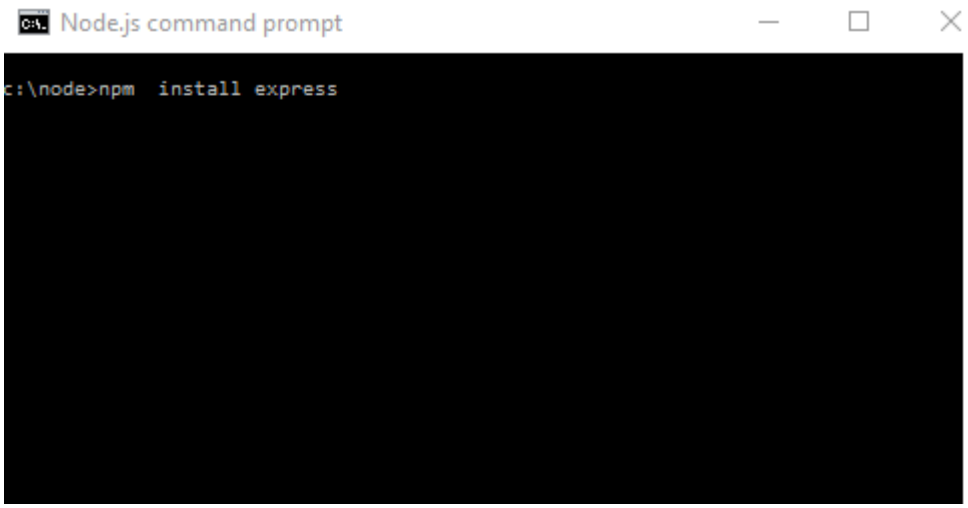
Kuva13. Yksinkertainen koodiesimerkki http-palvelimesta joka toimii Nodessa.

4.2.1 Node Package Manager

Node.js sisältää monipuolisen Node Package Managerin paketinhallintaohjelmiston, sen kautta verkkosovellukseen voidaan lisätä (kuva 14), päivittää tai poistaa Nodelle luotuja ohjelmapaketteja sekä hallita verkkosovelluksen sisältämien ohjelmistojen riippuvuuksia.

NPM:n verkkosivuilta löytyy varsin kattavasti Nodelle tehtyjä sovelluksia ja kirjastoja. Paketinhallinnan kautta voidaan paketoida myös omia sovelluksia tai jakaa niitä muiden Node.JS käyttäjien kesken.

NPM:n sovellukset noudattavat semanttista versionumerointia, eli versiointi ilmaistaan esimerkiksi seuraavasti: 1.5.2. Versionumerointi luetaan vasemmalta oikealle. Ensimmäinen numero ilmoittaa varsinaisen versionumeron ja sillä yleensä ilmoitetaan, että sovellukseen on tullut uusia ominaisuuksia. Keskimmäinen numero ilmoittaa, mikäli sovellukseen on tullut laajempia muutoksia ja viimeisellä numerolla ilmaistaan sovelluksen pienemmät korjaukset (Alicea 2015).



Kuva14. Tässä asennetaan express-lisäsovellus Nodeen.

4.3 Play! Framework

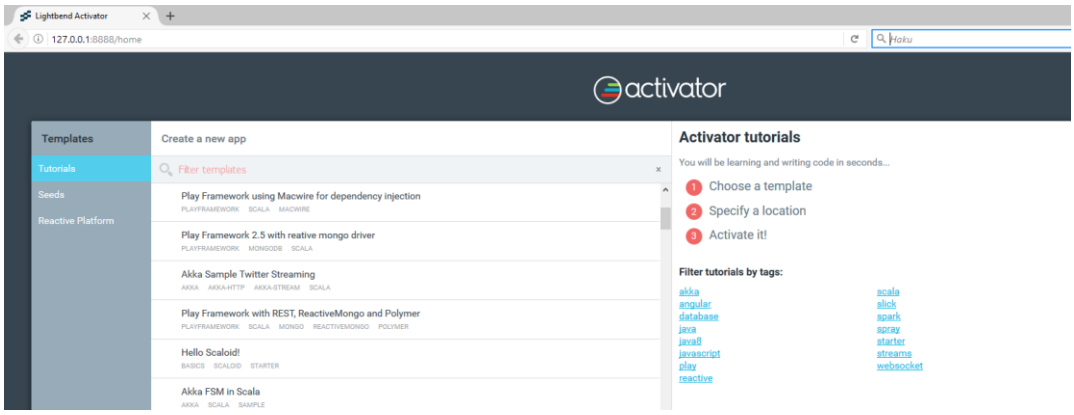
Play! Framework on Scala-ohjelmointikielellä kirjoitettu verkkosovelluskehys ja sen vahvuuksia ovat mukana tulevat kehitystyökalut, joita ei tarvitse erikseen ladata. Käytännössä se on täysiverinen verkkosovelluskehys heti asentamisen jälkeen ja sillä voidaan luoda Java tai Scala-pohjaisia verkkosovelluksia. Sovellukset noudattavat tarkasti MVC-mallinnus-arkkitehtuuria, verkkosovelluskehys tukee luonnollisesti myös Javalle luotuja kirjastoja. Play! Frameworkissa käytetään SBT-ohjelmaa verkkosovelluksen rakennukseen, kyseessä on siis Mavenin kaltainen projektin ylläpitoon liittyvä työkalu, joka huolehtii myös sovelluksen riippuvuuksien hallinnasta (Gontovnikas 2016).

Verkkosovelluskehys pitää sisällään myös oman http-palvelimen CakePHP-verkkosovelluskehyksen tavoin, joten sitä ei tarvitse erikseen määritellä. Oletuksena Play Frameworkin hallintasivusto toimii ***localhost***-osoitteella palvelimen portissa 8888 ja verkkosovelluksen kehitysympäristö portissa 9000.

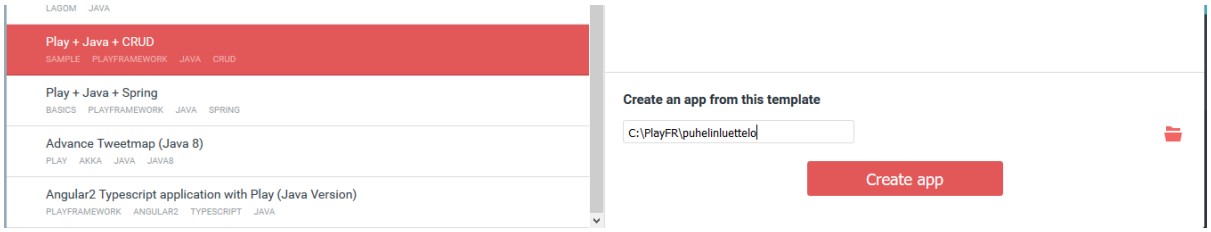
JUnit- testauksessa voidaan käyttää myös ”tekaistua” http-palvelinta, joka toimii erillisessä portissa. Näin saadaan testattua myös sovelluksen koodin toimivuus myös varsinaisella internet-sivustolla. Verkkosovelluskehyksessä on muutenkin tuettu hyvin JUnit-testausta. Play! Frameworkin kotisivuilla on todella laajat dokumentit, kuinka testata ja luoda toimiva sekä tietoturvallinen verkkosovellus.

Verkkosovelluskehyksen asennus sekä käyttöönotto tapahtuvat seuraavasti. Ensiksi ladataan pakattu asennustiedosto Play! Framework-yhteisön sivuilta. Asennuspaketit löytyvät MacOSX-, Linux ja Windows-käyttöjärjestelmille.

Tämän jälkeen ladattu tiedosto puretaan käyttäjän valitsemaan kansioon. Tiedoston purkamisen jälkeen kansiossa on bin-alihakemisto joka sisältää ***activator***-ohjelman. Kyseinen ohjelma käynnistetään ja se avaa internet selaimessa sivuston (Kuva 15) jonka kautta verkkosovelluskehittäjä voi valita valmiin mallipohjan omalle verkkosovellusprojektille tai tuoda aiemmin luodun projektin nykyiseen verkkosovelluskehykseen. Mikäli luodaan uutta projektia, sivustolta kannattaa etsiä sopiva mallipohja projektia varten. Näin voidaan säästää hieman ohjelmoinnissa aikaa eikä uutta projektia tarvitse aloittaa täysin alusta (Kuva16).



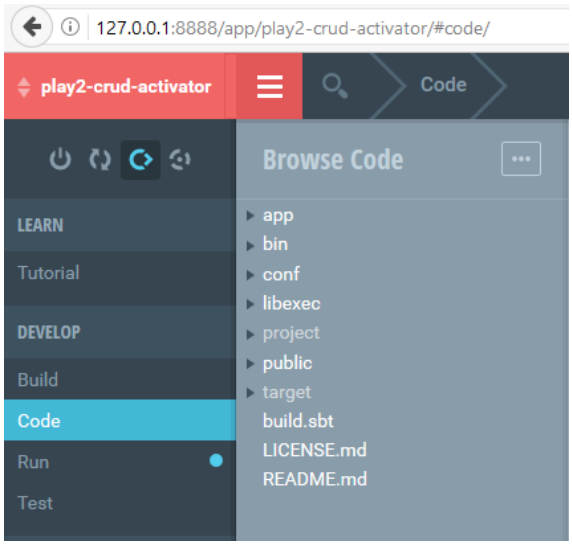
Kuva15.Activator-ohjelman käynnistää asetusverkkosivuston.



Kuva16.Esimerkki uuden verkkosovelluksen luomisesta mallipohjan avulla.

Kun Play! Framework luo uuden verkkosovellus-projektin, se avaa uuden verkkosivun, jossa on verkkosovelluksen hallintapaneeli. Kyseisestä hallintapaneelista voidaan kätevästi kääntää luodun sovelluksen koodia, suorittaa viimeisemmät muutokset reaaliajassa tai testata ohjelmakoodin toimivuus (Kuva 17). Hallintapaneelin avulla projekti voidaan myös kääntää suoraan IDE-kehitysympäristöön sopivaksi esimerkiksi Eclipselle tai IntelliJ-kehitysalustalle.

Play! Frameworkissa projektin kansiorakenne on varsin yksinkertainen (kuva 17). Verkkosovelluksen ***app***-kansiossa sijaitsevat toiminnan kannalta tärkeät kontrollerit, mallit ja näkymät. Sovelluksen konfiguraatiotiedostot ja sivuston reititysasetukset sijaitsevat puolestaan ***conf***-kansiossa. Verkkosovelluksen varsinaisena kotikansiona toimii ***public***-kansio, sinne lisätään myös sovelluksen kuva ja JavaScript tiedostot.



Kuva17.Hallintapaneeli sekä projektin kansiorakenne.

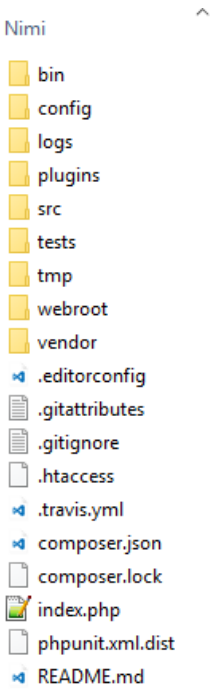
4.4 CakePHP

CakePHP on Cake Software Foundation kehittämä PHP-pohjainen verkkosovelluskehys, sen ensimmäinen versio julkaistiin vuonna 2005. Opinnäytetyötä kirjoittaessa uusin versio oli 3.3. CakePHP on yksi yleisimmistä PHP-sovelluskehyksistä ja se on suunniteltu siten, että verkkosovelluskehittäjät pystyisivät varsin nopeasti luomaan uusia verkkosovelluksia. Se noudattaa MVC-suunnitteluarkkitehtuuria Scala-pohjaisen Play! Frameworkin tavoin. (CakePHP 2016).

CakePHP on etenkin edukseen laajoissa ja perinteisissä verkkosovellusprojekteissa, joissa sovelluksen ylläpito, tietoturvallisuus ja hallittavuus ovat tärkeimpiä prioriteetteja. CakePHP tukee oletuksena Bootstrap-sovelluskehystä sekä JavaScript-skriptikieltä. Se tukee myös ORM-mallinnusta, joten sen avulla tietokannan käsittelyä voidaan suorittaa suoraan olio-orientoineesti. Yrityksen sivuilla oleva Cookbook-dokumentaatio tarjoaa CakePHP:stä aloittajalle tai kokeneelle ohjelmoijalle varsin laajan dokumentaation alkaen alkuasennuksesta suoraan käyttöönottoon tai jopa vaativien verkkosovellusten tekoon. (CakePHP 2016).

CakePHP:n asennus sekä käyttöönotto on tehty varsin helpoksi ja vaivattomaksi. CakePHP:n projektin asennus tapahtuu komentoriviltä Composer-ohjelman avulla, joka on PHP:lle suunniteltu riippuvuuksienhallintaohjelma. Asennuksen valmistuttua suoritetaan projektin ***bin***-kansiosta (Kuva 18) Cake http-palvelinsovellus, joka käynnistää verkkosovelluksen kehitystilassa, lopuksi määritellään vielä verkkosovelluksen tietokanta-asetukset.

Tämän jälkeen voidaan lähteä ohjelmoimaan varsinaista verkkosovellusta. ***src***-kansiossa (kuva 18) sijaitsevat verkkosovelluksen kooditiedostot. Näitä ovat mm. kontrollerit, mallit sekä näkymät jotka vastaavat sovelluksen toimintalogiikasta, mallinnuksesta sekä ulkoasusta. ***config***-kansiossa määritellään verkkosovelluksen asetukset ja verkkosivujen reititykset. Verkkosovelluksen kotikansiona toimii webroot, joka sisältää HTML, PHP ja JavaScript dokumentit. Kun verkkosovellus on saatu valmiiksi, niin sovelluksen status muutetaan lopuksi kehitystilasta tuotantotilaan.



Kuva 18. Projektin kansiorakenne heti asennuksen jälkeen.

4.5 Smarty

Smarty on PHP:lle suunniteltu pienimuotoinen mallinnejärjestelmä, jonka avulla PHP-koodi voidaan erottaa HTML tai XHTML-dokumentista. Se soveltuu erityisesti pieniin PHP-verkkosovelluksiin. Smartyn asennukseen riittää pakattu lähdekoodipaketti, jonka saa ladattua Smarty-yhteisön nettisivuilta. Lähdekoodit puretaan latauksen jälkeen verkkosovelluksen juurikansion sisälle erilliseen alihakemistoon. (Talvivaara 2016).

Smartyn käyttöönotto tapahtuu lisäämällä kooditiedostoon ***require_once*** attribuutin, joka viittaa ***Smarty.class.php***-tiedostoon (Kuva 19). Tämä on Smartyn luokkatiedosto ja se sijaitsee samassa alihakemistossa muiden Smarty tiedostojen kanssa. Smarty-luokasta luodaan uusi instanssi. Samalla määritellään sapluuna-kansio, josta verkkosovelluksen TPL-mallinnedokumentit ladataan. Tässä tapauksessa sapluuna-kansio on määritelty ***views***-nimellä. Lopuksi määritellään väliaikainen kansio jossa Smarty voi hoitaa ohjelmakoodin kääntämisen Kuva (19).

```php
<?php

require_once 'Smarty/libs/Smarty.class.php';
$smarty= new Smarty();
$smarty->template_dir = 'views';
$smarty->compile_dir = 'tmp';
```

Kuva 19. Esimerkki PHP-koodista, missä Smarty otetaan käyttöön.

Kun Smarty on määritelty verkkosovellukseen, sen ominaisuuksia voidaan käyttää. ***Assign***-funktiolla asetetaan mallinne tiedostossa näytettävät muuttujat ja display-funktiolla kutsutaan sapluuna kansiossa olevia TPL-dokumentteja. Ne ovat käytännössä HTML-dokumentteja, joiden tiedostopääte on vain muutettu tpl-päätteiseksi Kuva (20).

```
$smarty->assign('listAccounts', ListAccounts());
$smarty->display('index.tpl');
```

Kuva 20.Tässä esimerkissä asetetaan muuttuja sekä mallinne-tiedosto.

TPL-mallinnedokumentissa riittää vain viittaus PHP-koodin muuttujiin (Kuva21), ne on määritelty PHP-tiedostossa Smarty ***assign***-funktion avulla. Mallinnetiedostossa muuttujien nimet merkitään aina aaltosulkeiden sisään. Koska verkkosovellus suorittaa aina ensiksi toiminnallisen PHP-koodin, se ottaa samalla Smartyn ominaisuudet käyttöön. Lopuksi Smarty yhdistää TPL-mallinnetiedoston ja PHP-koodin toimivaksi kokonaisuudeksi. Smarty yhteisön sivuilla on varsin kattavat dokumentit, joista löytyy lisätietoa Smartyn muista ominaisuuksista. Smarty kuitenkin tukee myös oletuksena valinta sekä toistorakenteita.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Smarty-malli</title>
</head>
<body>
{$listAccounts}
</body>
</html>
```

Kuva 21.Tässä viitataan muuttujaan, joka sijaitsee PHP-tiedossa.

5 Tietokannat

5.1 MariaDB

MariaDB on avoimen lähdekoodin relaatiotietokanta-järjestelmä. Se perustuu vahvasti nykyisin Oraclen omistamaan MySQL:n tietokantajärjestelmään. Sitä kehittää MariaDB Foundation ja sen pääkehittäjiin kuuluu suomalainen Michael Widenius, joka oli alkujaan kehittämässä myös mySQL-tietokantajärjestelmää.

MariaDB on kehittyneempi tietokantajärjestelmä Oraclen MySQL:stä, ja se tukee monipuolisemmin tietokantamoottoreita sekä tietokannan optimointia. MariaDB:n versiointi vastaa mySQL:n versionumerointia, näin varmistetaan yhteensopivuus tietokantajärjestelmien välillä. Käytännössä tämä tarkoittaa sitä, että esimerkiksi MariaDB versio 5.3 on täysin yhteensopiva MySQL 5.3 version kanssa. Tämä mahdollistaa muun muassa sen, että MariaDB:llä voidaan korvata vastaava MySQL-pohjainen tietokanta. (MariaDB Foundation 2016).

5.2 MongoDB

MongoDB on dokumenttipohjainen tietokanta-järjestelmä, ja se kuuluu NOSQL-pohjaisiin tietokantoihin. MongoDB:ssä data tallennetaan kokoelmiin dokumentteina, periaatteessa kokoelmat vastaavat relaatiotietokantojen taulukoita. Dokumentit tallennetaan BSON-muodossa, joka on käytännössä binäärinen JSON-tiedonvälitystekniikka ja se tukee yleisempiä primitiivitietotyyppejä sekä JavaScript koodia. MongoDB:n ja dokumentti-pohjaisten tietokantojen etuna on skaalaavuus verrattuna perinteisiin reaalitietokantoihin sekä ne pystyvät käsittelevään isoja tietomääriä varsin nopeasti ja tehokkaasti. (Alicea 2015).

6 Tulokset

Opinnäytetyön aiheena olevan tietojärjestelmän kehityskohteita arvioitiin monesta eri näkökulmasta, mutta kehityskohteiden arvioinnissa keskityttiin erityisesti tietoturvaan, käytettävyyteen ja ylläpitoon. Alustavasti tiedettiin, että nämä asiat vaativat eniten korjaustoimenpiteitä. Nykyisen tietojärjestelmän kehityskohteita arvioitiin siitä näkökulmasta, millainen on hyvä tietojärjestelmä ja kuinka se olisi mahdollisimman käyttäjäystävällinen.

Kehityskohteiden arvioinnissa ei käytetty mittareita, vaan arvioinnin painoarvo oli enemmän pohdinnassa ja uusien havaintojen sekä näkökulmien löytämisessä. Kehityskohteita olisi voinut myös mitata, näin olisi saatu tieteellistä tietoa kehityskohteista. Ongelmana oli kuitenkin löytää sopivat mittarit ja mittauskohteet varsinaiseen arviointiin. Pohtimisen ja uusien havaintojen löytämisessä on se etu, että joutuu todella miettimään toimivia ratkaisuita kehityskohteisiin ja ajattelemaan ajoittain kriittisestikin. Tällöin avautuu uusi oivaltavia näkökulmia asiaan. Kehityskohteiden tarkastelu toi kuitenkin pääpiirteissään selkeän näkemyksen mihin suuntaan tietojärjestelmää on lähdettävä kehittämään ja millä se jatkossa toteutetaan.

Saatujen tuloksien perusteella nykyinen tietojärjestelmä olisi järkevintä korvata Play! Framework tai Node.js-verkkosovelluskehyksillä, koska edellä mainitut verkkosovelluskehykset edustavat nykyaikaista verkkosovelluskehitystä ja ne tukevat reaaliaikaista sekä asynkronista ohjelmakoodin suoritusta. Kolmantena vaihtoehtona ollut CakePHP-verkkosovelluskehys ei soveltunut tähän projektiin yhtä hyvin kuin kaksi edellä mainittua, koska painoarvo oli enemmän sovelluksen reaaliaikaisuudessa ja sen ketteryydessä. CakePHP on suunniteltu lähinnä isompien verkkosovellusten tekoon, myös sen teknologia perustuu enemmän perinteiseen PHP-pohjaiseen palvelin-asiakas-verkkosovellusratkaisuun

Kaksi ensiksi mainittua verkkosovelluskehystä olivat myös käytännön kokeilujen perusteella helposti asennettavissa ja toimivat varsin hyvin erilaisilla käyttöjärjestelmäalustoilla. Molempien sovelluskehysten dokumentaatiot olivat myös varsin kattavat, joten tarvittaessa niistä voi hakea lisätietoja verkkosovelluksen toteutukseen. Kummassakin verkkosovelluskehyksessä oli myös sisäinen paketinhallinta, jonka avulla verkkosovelluskehittäjä pystyy lisäämään verkkosovellukseen lisäosia tai kirjasto-ohjelmia.

Mainittujen verkkosovellusten avulla voidaan löytää jo valmiiksi kehitettyjä ratkaisuita luvussa kolme mainittuihin kehityskohteisiin, joita olivat mm. tietoturvallisuus, laajennettavuuteen, käyttöliittymään tai interaktiivisuuteen liittyvät ratkaisut. Verkkosovelluksissa voidaan käyttää myös NO-SQL-pohjaisia tietokantoja, joka tarjoavat tuen ison datamäärän tallennukseen.

7 Pohdintaa opinnäytetyöstä

Opinnäytetyö oli onnistunut ja se saavutti ne vaaditut tavoitteet, jotka sille oli asetettu. Työ oli varsin antoisa ja sen kautta opin paljon verkkosovelluskehityksestä. Sain myös hyvän näkemyksen siitä, kuinka opinnäytetyön aiheena olevaa tietojärjestelmää voidaan kehittää.

Opinnäytetyön aikataulu sekä suunnitelma muuttuivat merkittävästi, sillä alkuperäisessä suunnitelmassa oli myös toteuttaa samalla tietojärjestelmä, joka käyttäisi valmista verkkosovelluskehystä, näin verkkosovelluksen rakennusprosessia olisi voitu kuvata suoraan opinnäytetyössä. Opinnäytetyön aikataulun kanssa oli myös haasteita. Aikataulu sekä suunnitelmat muuttuivat kuitenkin lähinnä sen vuoksi, että sain uuden työpaikan ja näin opinnäytetyöhön käytettävä aika supistui suunnitellusta.

Olin suunnitellut opinnäytetyön ajankäytön niin, että työstän opinnäytetyötä aina päiväsaikaan, nyt se ei kuitenkaan onnistunut ja illasta ajankäyttö jäi lyhyeksi. Myös opinnäytetyön sisällön laajuus kasvoi sitä mukaa kuin omaksuin uutta tietoa ja tuntui siltä, että halusin rakentaa opinnäytetyöstä mahdollisimman monipuolisen kokonaisuuden.

Lähteet

Alicea, A. 2015. Learn and Understand NodeJS
https://www.udemy.com/understand-nodejs/learn/v4/overview uR8. 20.8.2016.

Apachefriends.2016. Xampp.
https://www.apachefriends.org/index.html uR8 5.12.2016

Apple. 2015.Model-View-Controller.
https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html uR8. 5.12.2016.

Korpela, J. 2013. CSS3 -uudet mahdollisuudet. Jyväskylä: Docendo.

Korpela, J. 2011. HTML5 -uudet ominaisuudet. Jyväskylä: Docendo.

Korpela, J. & Linjamaa, T. 2005. Web-suunnittelu. Jyväskylä: Docendo.

Kuha, J. 2007. Tehokas Java EE-sovellustuotanto. Jyväskylä: Docendo

Linjamaa, A. 2005. Web-ohjelmointi. Jyväskylä: Docendo.

MariaDB Foundation.2016. MariaDB versus MySQL - Features
https://mariadb.com/kb/en/mariadb/mariadb-vs-mysql-features/ uR8. 20.8.2016.

Percival, R. 2015a. The Complete Node JS Developer Course
https://www.udemy.com/the-complete-node-js-developer-course/learn/v4/overview uR8. 20.8.2016.

Percival, R. 2015b. The Complete Web Developer Course
https://www.udemy.com/complete-web-developer-course/learn/v4/overview uR8. 20.8.2016.

Rantala, A. 2005. Web-ohjelmointi.
Jyväskylä: Docendo.

Smith, D. & Negrino, T. 2007 JavaScript rakenna dynaamisia verkkosivuja.
Jyväskylä: Gummerus Kirjapaino.

Talvivaara, J. 2016. PHP-ohjelmoinnin jatkokurssi.
Karelia-ammattikorkeakoulu. 20.8.2016

Włodarczyk, A. 2014. JavaScript from Beginner to Expert
https://www.udemy.com/JavaScript-from-beginner-to-expert-bring-life-to-your-site/learn/v4/announcements uR8. 20.8.2016.

Westerholm, M. & Kyyppö, J. 2015. Java ohjelmointi. Liettua: Talentum.

W3C.2016. Facts About W3C.
https://www.w3.org/Consortium/facts#history uR8. 5.12.2016.

W3Schools.2016. jQuery Tutorial
http://www.w3schools.com/jquery/ uR8. 5.12.2016

W3Schools.2016. JSON - Introduction
http://www.w3schools.com/js/js_json_intro.asp uR8. 5.12.2016